Discrete Mathematics		
	Chapter 3	
Instructor: Bow-Yaw Wang		Spring 2004

Mathematical Reasoning, Induction, and Recursion

1. Proof Strategy

Sometimes, we may make a statement without knowing whether it is true or not. Such statements are called **conjectures**. When a conjecture is made, we can either prove it and make it a theorem. Or, we can find a **counterexample** to illustrate the conjecture is false.

Conjecture 1. *Let p_n be $(1+$ the product of the first n primes$)$. Then p_n is a prime for all n.*

For instance, $p_1 = 1 + 2 = 3$, $p_2 = 1 + 2 \cdot 3 = 7$, and $p_3 = 1 + 2 \cdot 3 \cdot 5 = 31$ are primes. Is it true for all n?

Example 1. *Find a counterexample to falsify Conjecture 1.*

Solution.$p_6 = 1 + 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 = 30031$. But $30031 = 59 \cdot 509$ is not a prime. □

Theorem 1. *(Fermat's Last Theorem) The equation*

$$x^n + y^n = z^n$$

has no solutions $x, y, z \in \mathbb{Z}$ with $xyz \neq 0$ whenever $n > 2$.

Remark. Notice that when $n = 2$, it is Pythagoras's Theorem.

Conjecture 2. *(Goldbach's Conjecture) For all even integer n and $n > 2$, n is the sum of two primes.*

2. Sequences and Summations

Definition 1. *A **sequence** is a function from $\mathbb{N}$ (or $\mathbb{Z}^+$) to a set. We use a_n to denote the image of n. a_n is called a **term** of the sequence. Sometimes, we write $\{a_n\}$ to describe the sequence.*

Definition 2. *A **string** is a sequence $\{a_n\}$ from $\mathbb{Z}^+$ to some alphabet. A string of **length** n is denoted by $a_1 a_2 \cdots a_n$. The **empty string** is denoted by λ.*

Example 2. *How can we produce the terms of a sequence if the first 5 terms are 3,1,4,1,5?*

*Solution.*There are infinitely many solutions. Consider the following function:

$$\begin{aligned} a_n \;=\; & 3 \cdot \frac{(n-2)(n-3)(n-4)(n-5)}{(1-2)(1-3)(1-4)(1-5)} + 1 \cdot \frac{(n-1)(n-3)(n-4)(n-5)}{(2-1)(2-3)(2-4)(2-5)} + \\ & 4 \cdot \frac{(n-1)(n-2)(n-4)(n-5)}{(3-1)(3-2)(3-4)(3-5)} + 1 \cdot \frac{(n-1)(n-2)(n-3)(n-5)}{(4-1)(4-2)(4-3)(4-5)} + \\ & 5 \cdot \frac{(n-1)(n-2)(n-3)(n-4)}{(5-1)(5-2)(5-3)(5-4)}. \end{aligned}$$

Then $a_1 = 3$, $a_2 = 1$, etc. □

Figure 1 shows some useful summation formulae.

Sum	Closed Form
$\sum_{k=0}^{n} ar^k (r \neq 0)$	$\frac{ar^{n+1}-a}{r-1}, r \neq 1$
$\sum_{k=1}^{n} k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k, \lvert x\rvert < 1$	$\frac{1}{1-x}$
$\sum_{k=0}^{\infty} kx^{k-1}, \lvert x\rvert < 1$	$\frac{1}{(1-x)^2}$

Figure 1. Useful Summation Formulae

Example 3. *Prove*

$$\sum_{k=0}^{\infty} kx^{k-1}, |x| < 1 = \frac{1}{(1-x)^2}$$

Proof. Differentiate both sides of the equation

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}.$$

□

3. Mathematical Induction

Given the propositional $P(n)$ where $n \in \mathbb{N}$, a proof by **mathematical induction** is of the form:
BASIS STEP: The proposition $P(0)$ is shown to be true.
INDUCTIVE STEP: The implication $P(k) \to P(k+1)$ is shown to be true for every $k \in \mathbb{N}$.
In the inductive step, the statement $P(k)$ is called the **inductive hypothesis**.

Example 4. *Show that if S is a finite set with n elements, then S has 2^n subsets.*

Proof. Let $P(n)$ be the proposition "a set with n elements has 2^n subsets." We use mathematical induction to prove $P(n)$ is true for all n.

BASIS STEP: $P(0)$ is true. Clearly, the empty set has only one $(= 2^0)$ subset.

INDUCTIVE STEP: Assume $P(k)$ is true. We would like to show $P(k+1)$ is true. Let T be a set of $k+1$ elements. Without loss of generality, we may assume $T = \{a_0, a_1, \ldots, a_k\}$. Then $T' = \{a_0, a_1, \ldots, a_{k-1}\}$ is a set of k elements and $T = T' \cup \{a_k\}$. By inductive hypothesis, T' has 2^k subsets. Then any subset of T is either X or $\{a_k\} \cup X$ for some subset X of T'. Hence the number of subsets of $T = 2 \cdot 2^k = 2^{k+1}$. □

Given the propositional $P(n)$ where $n \in \mathbb{N}$, a proof by **second principle of mathematical induction** (or **strong induction**) is of the form:
BASIS STEP: The proposition $P(0)$ is shown to be true.
INDUCTIVE STEP: The implication $[P(0) \wedge P(1) \wedge \cdots \wedge P(k)] \to P(k+1)$ is shown to be true for every $k \in \mathbb{N}$.

Example 5. *Let $n \in \mathbb{N}$ and $n \geq 12$. Show that $n = 4i + 5j$ for some $i, j \in \mathbb{N}$.*

Proof. Let $P(n)$ be the proposition "$n = 4i + 5j$ for some $i, j \in \mathbb{N}$." We use strong induction to show that $P(n)$ holds for $n \geq 12$.

BASIS STEP: $P(12), P(13), P(14), P(15)$ are easy to verify. $12 = 4 \cdot 3$. $13 = 4 \cdot 2 + 5$. $14 = 4 + 5 \cdot 2$.

INDUCTIVE STEP: Let $k \geq 15$. Assume $P(j)$ is true for $12 \leq j \leq k$. Then $k + 1 = (k - 3) + 4$. By inductive hypothesis, $k - 3 = 4i + 5j$ for some $i, j \in \mathbb{N}$. Hence $k + 1 = 4(i + 1) + 5j$. □

Remark. Can you prove the statement for $n \geq 4$? If not, why?

Definition 3. *Let S be a set. We say S is **well-ordered** if there is a relation $<$ defined over S such that*

1. *For any $a, b \in S$, either $a = b$, $a < b$ or $b < a$.*
2. *For any $a, b, c \in S$ with $a < b$ and $b < c$, $a < c$.*
3. *every non-empty subset of S has a least element (in terms of $<$).*

We call (3) *the **well-ordering property***

For instance, $\mathbb{N}$ and $\mathbb{Z}^+$ are well-ordered. But $\mathbb{Z}$ and $\mathbb{Q}^+$ are not. (why?)

The validity of mathematical induction depends on the well-ordering property. Let us write the mathematical induction formally:

$$[P(0) \wedge (P(k) \to P(k+1))] \to \forall n P(n).$$

Suppose $\neg \forall n P(n)$. Let $S = \{m : \neg P(m)\}$. By our assumption, $S \neq \emptyset$. Hence S has a least element m_0 by the well-ordering property. If $m_0 = 0$, we're done. (why?) On the other hand, $m_0 > 0$. Hence $m_0 - 1 \geq 0$ and $P(m_0 - 1)$ is true by the choice of m_0. Therefore $P(m_0 - 1) \to P(m_0)$ is false. We conclude our proof.

4. Recursive Definitions and Structural Induction

Definition 4. *The **Fibonacci numbers**, $f_0, f_1, \ldots$ are defined by*

- $f_0 = 0$ *and* $f_1 = 1$*;*
- $f_n = f_{n-1} + f_{n-2}$ *for* $n \geq 2$*.*

Example 6. *Show that when $n \geq 3$, $f_n > \alpha^{n-2}$, where $\alpha = \frac{1+\sqrt{5}}{2}$.*

Proof. *BASIS STEP*: $\alpha \approx 1.6180 < 2 = f_3$ and $\alpha^2 = \frac{3+\sqrt{5}}{2} \approx 2.6180 < f_4$.

INDUCTIVE STEP: Assume $f_j > \alpha^{j-2}$ for all j with $3 \leq j \leq k$. We want to show $f_{k+1} > \alpha^{k-1}$. By inductive hypothesis, we have $f_{k-1} > \alpha^{k-3}$ and $f_k > \alpha^{k-2}$. $f_{k+1} = f_{k-1} + f_k > \alpha^{k-3} + \alpha^{k-2}$. But $\alpha^{k-3} + \alpha^{k-2} = \alpha^{k-3}(\alpha+1) = \alpha^{k-3}\frac{3+\sqrt{5}}{2} = \alpha^{k-3}\alpha^2 = \alpha^{k-1}$. Hence $f_{k+1} > \alpha^{k-1}$. □

Example 7. *Show that*

$$f_i = \frac{\phi^i - \bar{\phi}^i}{\sqrt{5}}$$

where $\phi = \frac{1+\sqrt{5}}{2}$ and $\bar{\phi} = \frac{1-\sqrt{5}}{2}$.

Proof. *BASIS STEP*: $i = 0$, $\frac{\phi^0 - \bar{\phi}^0}{\sqrt{5}} = 0 = f_0$. $i = 1$, $\frac{\phi - \bar{\phi}}{\sqrt{5}} = 1 = f_1$.

INDUCTIVE STEP: Assume the equation holds up to k. Note that $\phi^2 = \frac{3+\sqrt{5}}{2} = \phi + 1$ and $\bar{\phi}^2 = \frac{3-\sqrt{5}}{2} = \bar{\phi} + 1$. We have

$$\begin{aligned}
f_{k+1} &= f_k + f_{k-1} \\
&= \frac{\phi^k - \bar{\phi}^k}{\sqrt{5}} + \frac{\phi^{k-1} - \bar{\phi}^{k-1}}{\sqrt{5}} \\
&= \frac{\phi^k + \phi^{k-1}}{\sqrt{5}} - \frac{\bar{\phi}^k + \bar{\phi}^{k-1}}{\sqrt{5}} \\
&= \frac{\phi^{k-1}(\phi+1)}{\sqrt{5}} - \frac{\bar{\phi}^{k-1}(\bar{\phi}+1)}{\sqrt{5}} \\
&= \frac{\phi^{k-1}\phi^2}{\sqrt{5}} - \frac{\bar{\phi}^{k-1}\bar{\phi}^2}{\sqrt{5}} \\
&= \frac{\phi^{k+1}}{\sqrt{5}} - \frac{\bar{\phi}^{k+1}}{\sqrt{5}} \\
&= \frac{\phi^{k+1} - \bar{\phi}^{k+1}}{\sqrt{5}}
\end{aligned}$$

□

Theorem 2. *(Lamé's Theorem) Let $a, b \in \mathbb{Z}^+$ with $a \geq b$. Then the number of divisions used by the Euclidean algorithm to find $\gcd(a,b)$ is less than or equal to five times the number of decimal digits in b.*

Proof. Let $a = r_0$ and $b = r_1$. The following sequence of equations is obtained in the Euclidean algorithm:

$$\begin{array}{rcll}
r_0 &=& r_1 q_1 + r_2 & 0 \leq r_2 < r_1 \\
r_1 &=& r_2 q_2 + r_3 & 0 \leq r_3 < r_2 \\
 & \cdot & & \\
 & \cdot & & \\
 & \cdot & & \\
r_{n-2} &=& r_{n-1} q_{n-1} + r_n & 0 \leq r_n < r_{n-1} \\
r_{n-1} &=& r_n q_n &
\end{array}$$

Note that $q_i \geq 1$ for all i and $q_n \geq 2$ since $r_n < r_{n-1}$. Therefore,

$$\begin{array}{rcccccl}
r_n & \geq & 1 & & & = & f_2 \\
r_{n-1} & \geq & 2r_n & \geq & 2f_2 & = & f_3 \\
r_{n-2} & \geq & r_{n-1}+r_n & \geq & f_3+f_2 & = & f_4 \\
 & \cdot & & & & & \\
 & \cdot & & & & & \\
 & \cdot & & & & & \\
r_2 & \geq & r_3+r_4 & \geq & f_{n-1}+f_{n-2} & = & f_n \\
r_1 = b & \geq & r_2+r_3 & \geq & f_n+f_{n-1} & = & f_{n+1}
\end{array}$$

Hence, if n divisions are needed by the Euclidean algorithm, then $b \geq f_{n+1}$. By Example 6, we have $f_{n+1} > \alpha^{n-1}$ for $n > 2$ where $\alpha = \frac{1+\sqrt{5}}{2}$. Hence $b > \alpha^{n-1}$. Now

$$\log b > (n-1)\log\alpha \approx (n-1)0.2090 > (n-1)/5.$$

Hence $n < 1 + 5\log b$. Let k be the number of decimal digits of b, $\log b < k$. Thus $n < 1 + 5k$. And $n \leq 5k$. □

Remark. The number of decimal digits of b is equal to $\lfloor \log b \rfloor + 1$. Hence the Euclidean algorithm uses $O(\log b)$ operations.

In addition to functions, we can also define other entities recursively.

Definition 5. *The set Σ^* of (finite) string over Σ is defined by*

- *$\lambda \in \Sigma^*$, the empty string;*
- *If $\omega \in \Sigma^*$ and $x \in \Sigma$, then $\omega x \in \Sigma^*$.*

Operations over string can also be defined recursively.

Definition 6. *The **concatenation** of two strings ω_0, ω_1 over Σ^*, write $\omega_0 \cdot \omega_1$, is defined by*

- *$\omega_0 \cdot \lambda = \omega_0$;*
- *$\omega_0 \cdot (\omega_1' x) = (\omega_0 \cdot \omega_1')x$ where $\omega_1 = \omega_1' x$.*

Definition 7. *A **rooted tree** is defined as follows.*

- *A single vertex v is a rooted tree with root v;*
- *Suppose $T_0, T_1, \ldots, T_n$ are rooted trees with roots $r_0, r_1, \ldots, r_n$. Then the graph formed by a new vertex r with edges connecting $r_0, r_1, \ldots, r_n$ is a rooted tree with root r.*

Remark. Can you draw some rooted trees?

Definition 8. *An **extended binary trees** is defined as follows.*

- *The empty set is an extended binary tree without root;*
- *Let T_0 and T_1 are extended binary trees. Then the graph formed by a new vertex r with edges connecting roots of the **left subtree** T_0 and the **right subtree** T_1 is an extended binary tree, $T_0 \cdot T_1$, with root r.*

Remark. Can you draw some extended binary trees?

Definition 9. *A **full binary tree** is defined as follows.*

- *A single vertex v is a full binary tree with root v;*
- *Let T_0 and T_1 are full binary trees. Then the graph formed by a new vertex r with edges connecting roots of the **left subtree** T_0 and the **right subtree** T_1 is a full binary tree, $T_0 \cdot T_1$, with root r.*

Remark. Can you draw some full binary trees? What are the differences among rooted trees, extended binary trees and full binary trees?

We can also prove properties of recursively defined entities by induction.

Definition 10. *The **height**, $h(T)$, of a full binary tree T is defined as follows.*

- *The height of the full binary tree T consisting of only a single vertex r is 0;*
- *If T_0 and T_1 are full binary trees and $T = T_0 \cdot T_1$, $h(T) = 1 + \max(h(T_0), h(T_1))$.*

Definition 11. *The **size**, $n(T)$, of a full binary tree T is defined as follows.*

- *The size of the full binary tree T consisting of only a single vertex r is 1;*
- *If T_0 and T_1 are full binary trees and $T = T_0 \cdot T_1$, $n(T) = 1 + n(T_0) + n(T_1)$.*

function *recursive-fibonacci*$(n : \mathbb{N})$
if $n = 0$ **then**
    return 0
else if $n = 1$ **then**
    return 1
else $\{\ n > 1\ \}$
    return *recursive-fibonacci*$(n-1)$ + *recursive-fibonacci*$(n-2)$

Figure 2. Recursive Algorithm for Fibonacci Numbers

function *recursive-gcd*$(a, b : \mathbb{N})$
if $a > b$ **then**
    return *recursive-gcd*(b, a)
else if $a = 0$ **then**
    return b
else $\{\ a \leq b$ and $a \neq 0\ \}$
    return *recursive-gcd*$(b \mod a, a)$

Figure 3. Recursive Algorithm for $\gcd(a, b)$

The proof of the following theorem is called **structural induction**. The induction proceeds by the structure of the entity.

Theorem 3. *If T is a full binary tree, then $n(T) \leq 2^{h(T)+1} - 1$.*

Proof. BASIS STEP: For the case where T consists of a single root, $n(T) = 1 = 2^{h(T)+1} - 1$.
INDUCTIVE STEP: Assume $n(T_0) \leq 2^{h(T_0)+1} - 1$ and $n(T_1) \leq 2^{h(T_1)+1} - 1$.

$$\begin{aligned} n(T) &= 1 + n(T_0) + n(T_1) \\ &\leq 1 + (2^{h(T_0)+1} - 1) + (2^{h(T_1)+1} - 1) \\ &\leq 2 \cdot \max(2^{h(T_0)+1}, 2^{h(T_1)+1}) - 1 \\ &= 2 \cdot 2^{\max(h(T_0), h(T_1))+1} - 1 \\ &= 2 \cdot 2^{h(T)} - 1 \\ &= 2^{h(T)+1} - 1 \end{aligned}$$

□

5. Recursive Algorithms

Definition 12. *An algorithm is called* ***recursive*** *if it solves a problem by reducing the problem to (simpler) instances of the same problem.*

Example 8. *Give a recursive algorithm for Fibonacci numbers.*

*Solution.*Figure 2 shows such an algorithm. □

Example 9. *Give a recursive algorithm for computing* $\gcd(a, b)$.

*Solution.*Figure 3 shows such an algorithm. □

We can also solve the sorting problem recursively. Figure 4 shows the merge sort algorithm.

Example 10. *Analyze the time complexity of the merge sort algorithm in Figure 4.*

*Solution.*First, observe that the number of operations used by *merge* on input of size n is $\Theta(n)$. Let $T(n)$ be the number of operations used by *mergesort* on input of size n. We can see $T(n) = 1 + n + 2T(n/2) + \Theta(n) = \Theta(n) + 2T(n/2)$. We can verify $T(n) = O(n \lg n)$. □

Remark. Can you find a sorting algorithm of time complexity $O(n)$?

```
procedure mergesort(L = a_0, ..., a_n)
if n > 0 then
    m := ⌊n/2⌋
    L_0 := a_0, ..., a_m
    L_1 := a_{m+1}, ..., a_n
    L := merge(mergesort(L_0), mergesort(L_1))
{ L is sorted }
```

```
procedure merge(L_0 = a_0, ..., a_m; L_1 = b_0, ... b_n)
{ a_0 ≤ a_1 ≤ ··· ≤ a_m and b_0 ≤ b_1 ≤ ··· ≤ b_n }
L := empty list
i := 0; j := 0
while i ≤ m ∧ j ≤ n do
    if a_i ≤ b_j then
        append a_i to L
        i := i + 1
    else { b_j < a_i }
        append b_j to L
        j := j + 1
od
if i ≤ m then
    { j > n }
    while i ≤ m do
        append a_i to L
        i := i + 1
    od
else
    { i > m }
    while j ≤ n do
        append b_j to L
        j := j + 1
    od
```

FIGURE 4. The Merge Sort Algorithm

Recursion and iteration are actually equivalent in the sense that each recursive algorithm can be transformed to an iterative algorithm and vice versa.

In functional languages such as Lisp and ML, it is very common (if not necessary) to write recursive functions. Programmers are able to analyze recursively defined functions by induction. Hence the correctness of program can be achieved.

6. Program Correctness

Definition 13. *A program segment S is said to be* ***partially correct with respect to*** *the* ***initial assertion*** *p and the* ***final assertion*** *q, write $p\{S\}q$, if whenever p is true for the input values of S and S terminates, then q is true for the output values of S.*

Remark. The notation $p\{S\}q$ is known as **Hoare triple** after Tony Hoare.

With Hoare triples, it is possible to define the *semantics* of programming languages. Here, we give some examples:

Consider the inference rules:

$$\frac{p\{S_0\}q \qquad r\{S_1\}t \qquad q \to r}{p\{S_0; S_1\}t}$$

and

$$\frac{(p \wedge c)\{S_0\}q \quad (p \wedge \neg c)\{S_1\}q}{p\{\textbf{if } c \textbf{ then } S_0 \textbf{ else } S_1\}q}$$

(1) **if** $x < 0$ **then**
(2) $\quad abs := -x$
(3) **else**
(4) $\quad abs := x$

FIGURE 5. A Program Segment for Computing $|x|$

Example 11. *Verify the program segment in Figure 5 with respect to the initial assertion* $\boldsymbol{T}$ *and final assertion* $abs = |x|$.

Proof. We have $(\mathbf{T} \wedge x < 0)\{abs := -x\}(abs = |x|)$ and $(\mathbf{T} \wedge x \not< 0)\{abs := x\}(abs = |x|)$. By the inference rule, we conclude $abs = |x|$ at the end of the program segment. □

In addition to conditionals, we have inference rules for loops.

$$\frac{(p \wedge c)\{S\}p}{p\{\textbf{while } c \textbf{ do } S \textbf{ od}\}(\neg c \wedge p)}$$

Remark. Hoare triples do not guarantee the program segment to terminate. Therefore $p\{S\}\mathbf{F}$ when S does not terminate. Can you find a program segment S such that $p\{S\}\mathbf{F}$ can be inferred?

In practice, we do not write programs as Hoare triples. However, we can add assertions to help us catch bugs. If you write C (or C++) programs, please try to use `assert()` as often as possible. They can save you a lot of time!

The subject of understanding the meaning of programs is called **program semantics**. As we have seen in the example, Hoare triples can be used to specify the semantics of programs. We call them **axiomatic semantics**. There are other ways to define the semantics of programs. The semantics of programming languages is yet another interesting field of theoretical computer science.

Let A and B be two types (`int`, `float` etc). Intuitively, "A is a subtype of B" means "an expression of type A is usable wherever an expression of type B is." For instance, `int` is a subtype of `float`.

Let $A \prec B$ denote that A is a subtype of B. Hence `float` $\prec$ `double` and `int` $\prec$ `float`. It is also easy to see that $f \prec g$ for $f : C \to A$ and $g : C \to B$ with $A \prec B$.

Example 12. *Let* A, A', B, B' *be types with* $A \prec A'$ *and* $B \prec B'$. *Suppose* $f : A' \to B$ *and* $f' : A \to B'$. *Show* $f \prec f'$.

Solution. We want to argue that we can always replace f' by f. Consider any occurrence of f'. The argument $a \in A$ of f' is also of type A' for $A \prec A'$. Therefore a can be an argument of f for $f : A' \to B$. On the other hand, $f(a) \in B$. $f(a)$ is usable wherever $f'(a)$ is because $B \prec B'$. □

Remark. Can you write a C++ or Java program to demonstrate the example?